A Variable-metric Variant of the Karmarkar Algorithm for Linear Programming

J.E. Dennis, Jr.[1]

A.M. Morshedi[2]

and

Kathryn Turner[3]

Technical Report 86-13, June 1986

Revised July 1986, September 1986, and January 1987

[1]Mathematical Sciences Department, Rice University, Houston, Texas 77251-1892. Research sponsored by DOE DE-AS05-82ER13016, ARO DAAG-29-83-K-0035, AFOSR 85-0243.

[2]Shell Development Company, Westhollow Research Center, Houston, Texas 77001

[3]Mathematical Sciences Department, Rice University, Houston, Texas 77251-1892. Research sponsored by ARO DAAG-29-83-K-0035, AFOSR 85-0243, Shell Development Company.

Report Documentation Page			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.			
1. REPORT DATE **JAN 1987**	2. REPORT TYPE		3. DATES COVERED **00-00-1987 to 00-00-1987**
4. TITLE AND SUBTITLE **A Vairable-metric Variant of the Karmarkar Algorithm for Linear Programming**			5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)			5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Computational and Applied Mathematics Department ,Rice University,6100 Main Street MS 134,Houston,TX,77005-1892**			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT			
15. SUBJECT TERMS			

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**34**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

A Variable-metric Variant of the Karmarkar Algorithm for Linear Programming

J.E. Dennis, Jr.[1], A.M. Morshedi[2], and Kathryn Turner[3]

Technical Report 86-13, June 1986

Revised July 1986, September 1986, and January 1987

Abstract

The most time-consuming part of the Karmarkar algorithm for linear programming is the projection of a vector onto the nullspace of a matrix that changes at each iteration. We present a variant of the Karmarkar algorithm that uses standard variable-metric techniques in an innovative way to approximate this projection. In limited tests, this modification greatly reduces the number of matrix factorizations needed for the solution of linear programming problems.

Key words:

linear programming
Karmarkar algorithm
projective algorithm
variable-metric Karmarkar

Abbreviated title:

Variable-metric Karmarkar Algorithm

---

[1]Mathematical Sciences Department, Rice University, Houston, Texas 77251-1892. Research sponsored by DOE DE-AS05-82ER13016, ARO DAAG-29-83-K-0035, AFOSR 85-0243.

[2]Shell Development Company, Westhollow Research Center, Houston, Texas 77001

[3]Mathematical Sciences Department, Rice University, Houston, Texas 77251-1892. Research sponsored by ARO DAAG-29-83-K-0035, AFOSR 85-0243, Shell Development Company.

1. Introduction: the Karmarkar algorithm

Since the introduction of the polynomial time algorithm for linear programming by N. Karmarkar [1984], there has been considerable interest in its use, and in developing strategies for modifying the algorithm to achieve greater computational efficiency. We present a variable-metric variant of the Karmarkar algorithm. The algorithm as proposed by Karmarkar employs a projective transformation to map the feasible region of the linear program onto the intersection of a simplex with an affine space, and then solves the problem

$$\begin{array}{lrl} \text{minimize} & c^T x & \\ \text{subject to} & Ax & = 0 \\ & e^T x & = n \\ & x & \geq 0, \end{array}$$

where $c, x, e \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $e = (1, \cdots, 1)^T$. We follow the suggestion of Shanno and Marsten [1985] in using $e^T x = n$, rather than $e^T x = 1$, as used by Karmarkar [1984]. We transform a standard linear programming problem to the above form by commonly used strategies that do not involve a projective transformation. For completeness, we include these strategies in Appendix I. We shall also assume that the minimum value of the objective function $c^T x$ is zero, and that we have an initial feasible point $x_0 > 0$. A means of getting such an initial feasible point and of transforming a given objective function to an objective function whose minimum value is zero is also included in Appendix I.

At each step of the Karmarkar algorithm, the current iterate $(x_1, \cdots, x_n)^T$ is mapped to the center $(1, \cdots, 1)^T$ of the simplex by the projective transformation

$$\tilde{x} = \frac{n}{e^T D^{-1} x} D^{-1} x,$$

where $D = \operatorname{diag}(x_1, \cdots, x_n)$, and we use $\tilde{x}$ to denote the transformed variable.

A step is taken in the transformed space in the direction of the negative projected gradient of a linear function, and the new point in the untransformed space is found by applying the inverse of the above projective transformation (restricted to $\{x \in \mathbb{R}^n : x \geq 0,\, e^T x = n\}$),

$$x = \frac{n}{e^T D\tilde{x}} D\tilde{x} \,.$$

The Karmarkar algorithm is :

```
given an initial feasible point $x_0 > 0$
  begin
          $x = x_0$
          while $c^T x$ is too large
          do
                $D = \text{diag}\ (x_1, \cdots, x_n), \quad B = \begin{bmatrix} AD \\ e^T \end{bmatrix}$
                project $Dc$ onto the nullspace of $B$ :
                      $c_p = [I - B^T(BB^T)^{-1}B]Dc$
                $\tilde{x} = e - \alpha c_p$
                $x = \frac{n}{e^T D\tilde{x}} D\tilde{x}$
          end do
  end
```

In section 3 we discuss how the steplength parameter α is chosen at each step.

2. A variable-metric variant of the Karmarkar algorithm

The most time-consuming part of the Karmarkar algorithm is the projection of the vector Dc onto the nullspace of a matrix B that changes at each iteration. We propose to approximate the direction so obtained, using secant updates to the initial matrix. The updating strategies we have adapted for use in this context have been used very successfully in the solution of nonlinear equations and in the minimization of nonlinear functions. By employing updates we reduce the amount of work required at each step, since we bypass the need to obtain a matrix factorization at every step. We shall approximate the direction $c_p = [I-B^T(BB^T)^{-1}B]Dc$ by

$$\hat{c}_p = D^{-1}\hat{D}[I-\hat{B}^T(\hat{B}\hat{B}^T)^{-1}\hat{B}]\hat{D}^T c, \tag{2.1}$$

where

$$\hat{B} = \begin{bmatrix} A\hat{D} \\ e^T D^{-1}\hat{D} \end{bmatrix} = BD^{-1}\hat{D},$$

and $\hat{D}$ is a nonsingular approximation to D. The new iterate in the transformed space is given by

$$\tilde{x} = e - \alpha\hat{c}_p \ .$$

Before addressing the way we obtain the matrix $\hat{D}$, we note that any direction $\hat{c}_p$ computed in this manner is a feasible direction for the linear program and the negative of it is a descent direction for Karmarkar's potential function,

$$f(x) = \sum_{i=1}^{n} \log\frac{c^T x}{x_i} \ . \tag{2.2}$$

Feasibility of the direction $\hat{c}_p$ follows from the observation that $\hat{c}_p$ is in the nullspace of B. Karmarkar's proof of the convergence of his algorithm and his polynomial time bound are based on achieving at least a constant amount of reduction in the potential function (2.2) at each step. This potential function is transformable by the projective transformation to a function having the same form. When f is expressed in terms of the transformed variable, we have

$$f(x) = \sum_{i=1}^{n} \log \frac{c^T D\tilde{x}}{\tilde{x}_i} + constant,$$

so that in the transformed space, we may consider

$$\tilde{f}(\tilde{x}) = \sum_{i=1}^{n} \log \frac{c^T D\tilde{x}}{\tilde{x}_i}. \tag{2.3}$$

Now

$$\nabla_{\tilde{x}} \tilde{f}(\tilde{x}) = \frac{n}{c^T D\tilde{x}} Dc - \tilde{D}^{-1}e, \quad \text{where } \tilde{D} = \text{diag }(\tilde{x}_1, \cdots, \tilde{x}_n),$$

and

$$\nabla_{\tilde{x}} \tilde{f}(e) = \frac{n}{c^T x_c} Dc - e,$$

where x_c denotes the current iterate in the untransformed space. The direction $-\hat{c}_p$ is a non-ascent direction for the potential function in the transformed space since

$$\begin{aligned}\nabla_{\tilde{x}} \tilde{f}(e)^T(-\hat{c}_p) &= \left(-\frac{n}{c^T x_c} Dc + e\right)^T \hat{c}_p \\ &= -\frac{n}{c^T x_c} c^T \hat{D}[I - \hat{B}^T(\hat{B}\hat{B}^T)^{-1}\hat{B}]\hat{D}^T c \\ &\leq 0 .\end{aligned}$$

In order to show that the direction $-\hat{c}_p$ is a descent direction for the potential function, we note that $c_p \neq 0$, since there exists a step in the direction $-c_p$ that yields a reduction in the potential function, and that $-\hat{c}_p$ can fail to be a descent direction only if $\hat{D}^T c$ is orthogonal to the nullspace of $\hat{B}$. But if $\hat{D}^T c$ is orthogonal to the nullspace of $\hat{B}$, then there exists $y \in \mathbb{R}^m$ such that

$$\begin{aligned} \hat{B}^T y &= \hat{D}^T c \\ \hat{D}^T D^{-1} B^T y &= \hat{D}^T c, \text{ and since } \hat{D} \text{ is nonsingular}, \\ B^T y &= Dc, \end{aligned}$$

which contradicts the fact that $c_p \neq 0$.

Our modification of the Karmarkar algorithm requires a factorization of the initial matrix $(AD_0)(AD_0)^T$ only. Thereafter, as is the usual approach in computations for large problems, update vectors are stored, and the projection computation is done cheaply, requiring only solutions of linear systems from the initially factored matrix, scalar products, and sums of vectors.

We begin with $\hat{D}_0 = D_0$, a diagonal matrix whose diagonal elements are the components of the initial feasible point x_0. We may obtain rank-one secant updates to this matrix through the use of one of two nonlinear functions whose gradients and Hessians involve the matrix D. The first and simplest of these is a logarithmic barrier function, and the second is the potential function we have already defined. Let us first examine the logarithmic barrier function.

Applying the barrier transformation to

$$\begin{aligned} \min_x \quad & c^T x \\ \text{subject to} \quad & Ax = 0 \\ & e^T x = n \\ & x \geq 0 \end{aligned} \tag{2.4}$$

gives

$$\begin{aligned} \min_x \quad & F(x) = c^T x - \mu \sum_{i=1}^{n} \log x_i \\ \text{subject to} \quad & Ax = 0 \\ & e^T x = n, \end{aligned} \tag{2.5}$$

the inequality constraints having been replaced by a term in the objective function. It is well known (see, for example, Gill, Murray, Saunders, Tomlin, and

Wright, [1985]) that under mild hypotheses the solution to (2.5) converges to the solution of (2.4) as $\mu \rightarrow 0$. It is not our intention to solve problem (2.5) for any value of μ; we intend merely to use the logarithmic barrier function, whose gradient and Hessian involve the matrix D , in order to obtain an approximation to this matrix. We have

$$\nabla_x F(x) = c - \mu D^{-1} e,$$

where $D =$ diag $(x_1, \cdots, x_n)$ and $e = (1, \cdots, 1)^T$; and

$$\nabla_x^2 F(x) = \mu D^{-2}.$$

Analogous to the idea of a secant approximation to the derivative of a function of one variable as the ratio of the change in function values to the change in the independent variable, a secant approximation to a Hessian is obtained by requiring the new approximation to the Hessian acting on the step in the independent variable to produce the difference in gradient vectors at the new and previous points. Using the subscript + to refer to the new iterate and the subscript c to refer to the previous (current) iterate, we may write the secant equation based on the logarithmic barrier function as

$$\begin{aligned} \mu \hat{D}_+^{-2}(x_+ - x_c) &= (c - \mu D_+^{-1} e) - (c - \mu D_c^{-1} e) \\ \hat{D}_+^{-2}(x_+ - x_c) &= (D_c^{-1} - D_+^{-1})e. \end{aligned} \tag{2.6}$$

We compute the righthand side exactly. Letting y_b denote $(D_c^{-1} - D_+^{-1})e$ and, not wishing to require the approximation $\hat{D}$ to be symmetric, replacing $\hat{D}_+^2$ with $\hat{D}_+ \hat{D}_+^T$, we require the approximation to satisfy

$$(x_+ - x_c) = \hat{D}_+ \hat{D}_+^T y_b .$$

An approximation based on the potential function (2.2) requires a very similar secant equation to be satisfied. Recall that the potential function is

$$f(x) = \sum_{i=1}^{n} \log \frac{c^T x}{x_i},$$

so that

$$\nabla_x f(x) = \frac{n}{c^T x} c - D^{-1} e \ ,$$
$$\nabla_x^2 f(x) = D^{-2} - \frac{n}{(c^T x)^2} cc^T \ ,$$

and the secant equation is

$$\left(\hat{D}_+^{-2} - \frac{n}{(c^T x_+)^2} cc^T \right) (x_+ - x_c) = \left(\frac{n}{c^T x_+} c - D_+^{-1} e \right) - \left(\frac{n}{c^T x_c} c - D_c^{-1} e \right) .$$

Moving the second term on the lefthand side of the equation to the righthand side and collecting terms gives

$$\hat{D}_+^{-2} (x_+ - x_c) = (D_c^{-1} - D_+^{-1}) e - \frac{n (c^T x_c - c^T x_+)^2}{(c^T x_+)^2 (c^T x_c)} c \ . \tag{2.7}$$

Now letting y_p denote the righthand side of the above equation, and again replacing $\hat{D}_+^2$ with $\hat{D}_+ \hat{D}_+^T$, we require the approximation to satisfy

$$(x_+ - x_c) = \hat{D}_+ \hat{D}_+^T y_p \ .$$

Recall that the matrix we are trying to approximate is a known diagonal matrix. A third and even simpler alternative requirement that can be imposed on the approximation is that it satisfy

$$\hat{D}_+^{-2} (x_+ - x_c) = D_+^{-2} (x_+ - x_c) \ . \tag{2.8}$$

We shall use y_t to denote $D_+^{-2} (x_+ - x_c)$ and express this requirement as

$$(x_+ - x_c) = \hat{D}_+ \hat{D}_+^T y_t \ .$$

To obtain an updating formula based on the barrier function or on the potential function or on the true matrix D_+ (y denotes either y_b or y_p or y_t,the righthand side of equation (2.6) or (2.7) or (2.8), respectively), we follow Dennis and Schnabel [1983]. For completeness, we include the development of the updating formula here. We express the requirement $(x_+ - x_c) = \hat{D}_+ \hat{D}_+^T y$ as

$$\begin{aligned}(x_+ - x_c) &= \hat{D}_+ v \\ v &= \hat{D}_+^T y\end{aligned} \tag{2.9}$$

for some vector $v \in \mathbb{R}^n$. Now we use the Broyden updating formula to obtain $\hat{D}_+$ as the closest matrix to $\hat{D}_c$ consistent with satisfying the first part of (2.9) :

$$\hat{D}_+ = \hat{D}_c + \frac{(x_+ - x_c - \hat{D}_c v) v^T}{v^T v} . \tag{2.10}$$

The second part of (2.9) requires that

$$v = \hat{D}_+^T y = \hat{D}_c^T y + \frac{(x_+ - x_c - \hat{D}_c v)^T y}{v^T v} v , \tag{2.11}$$

and this can be satisfied only if

$$v = \kappa \hat{D}_c^T y \tag{2.12}$$

for some $\kappa \in \mathbb{R}$. Now substituting (2.12) into (2.11) and simplifying gives

$$\kappa^2 = \frac{(x_+ - x_c)^T y}{y^T \hat{D}_c \hat{D}_c^T y} .$$

Assuming $(x_+ - x_c)^T y > 0$, we choose the positive root, so that

$$v = \left(\frac{(x_+ - x_c)^T y}{(\hat{D}_c^T y)^T (\hat{D}_c^T y)} \right)^{1/2} \hat{D}_c^T y . \tag{2.13}$$

For the update based on the true D , $y_t = D_+^{-2}(x_+ - x_c)$, so that

$$(x_+ - x_c)^T y_t = (x_+ - x_c)^T D_+^{-2} (x_+ - x_c) > 0 .$$

For the update based on the barrier function, $y_b = (D_c^{-1} - D_+^{-1})e$, and we have

$$(x_+ - x_c)^T y_b = (x_+ - x_c)^T D_c^{-1} D_+^{-1} (x_+ - x_c) > 0 .$$

For the update based on the potential function,

$y_p = (D_c^{-1} - D_+^{-1})e - \dfrac{n(c^T x_c - c^T x_+)^2}{(c^T x_+)^2 (c^T x_c)} c$, so that

$$(x_+ - x_c)^T y_p = (x_+ - x_c)^T D_c^{-1} D_+^{-1} (x_+ - x_c) - \frac{n[c^T(x_+ - x_c)]^3}{(c^T x_+)^2 (c^T x_c)} ,$$

which is not guaranteed to be positive, but is certainly positive if $c^T x_+ < c^T x_c$. If this updating strategy is being used and the above is not positive at some step, we choose to use the update based on the true D , rather than reject the update altogether. Thus far in our computational testing, the situation in which $(x_+ - x_c)^T y_p \leq 0$ has not arisen.

The updating formula given by equation (2.10) was obtained as a least-change secant update for approximating D, the matrix we are interested in, making use of nonlinear functions whose gradients and Hessians involve the matrix D^{-1}. We have also explored the use of the updating strategy that changes the approximation to the Hessian of the nonlinear function as little as possible (rather than changing the approximation to the inverse of the Hessian as little as possible) consistent with having the approximation satisfy the secant equation (2.6) or (2.7) or the condition (2.8). Using this approach, the approximation is required to satisfy

$$\hat{D}_+^{-1} \hat{D}_+^{-T} (x_+ - x_c) = y ,$$

and analogous to (2.9), we express this requirement as

$$\begin{aligned} y &= \hat{D}_+^{-1} v \\ v &= \hat{D}_+^{-T} (x_+ - x_c) \end{aligned} \tag{2.14}$$

for some vector $v \in \mathbb{R}^n$. Now the closest matrix $\hat{D}_+^{-1}$ to $\hat{D}_c^{-1}$ satisfying the first part of (2.14) is

$$\hat{D}_+^{-1} = \hat{D}_c^{-1} + \frac{(y - \hat{D}_c^{-1} v) v^T}{v^T v} , \tag{2.15}$$

and from the second part of (2.14) we obtain

$$v = \left[\frac{y^T(x_+ - x_c)}{(\hat{D}_c^{-T}(x_+ - x_c))^T(\hat{D}_c^{-T}(x_+ - x_c))} \right]^{\frac{1}{2}} \hat{D}_c^{-T}(x_+ - x_c) \,. \tag{2.16}$$

Using the Sherman-Morrison-Woodbury formula, (2.15) is equivalent to

$$\hat{D}_+ = \hat{D}_c + \frac{(v - \hat{D}_c y)(x_+ - x_c)^T}{y^T(x_+ - x_c)} \,. \tag{2.17}$$

Use of the update given by (2.17) requires more computation and more storage than is required if the update given by (2.10) is used, as we shall discuss in section 3. Furthermore, numerical experience thus far has indicated better performance is achieved using the update given by (2.10). The modified algorithm will be referred to as the DMT-Karmarkar algorithm.

3. Computational issues in the DMT-Karmarkar algorithm

3.1 Update vectors

The computation of $\hat{c}_p$ requires the solution of a linear system with coefficient matrix $\hat{B}\hat{B}^T$. Since

$$\hat{B}\hat{B}^T = \begin{bmatrix} M & a \\ a^T & \sigma \end{bmatrix},$$

where $M = (A\hat{D})(A\hat{D})^T$, $a = A\hat{D}\hat{D}^T D^{-1}e$, and $\sigma = e^T D^{-1}\hat{D}\hat{D}^T D^{-1}e$, we need only be concerned with solving a linear system with coefficient matrix $(A\hat{D})(A\hat{D})^T$. We first discuss the computation of $\hat{c}_p$ using the update to $\hat{D}$ given by (2.10) and (2.13). Referring to the updating formula (2.10),

$$A\hat{D}_+ = A\hat{D}_c - \frac{(A\hat{D}_c v)v^T}{v^T v} = A\hat{D}_c \left[I - \frac{vv^T}{v^T v} \right] .$$

Hence,

$$(A\hat{D}_+)(A\hat{D}_+)^T = A\hat{D}_c\left(I - \frac{vv^T}{v^Tv}\right)(A\hat{D}_c)^T$$

$$= (A\hat{D}_c)(A\hat{D}_c)^T - \frac{(A\hat{D}_cv)(A\hat{D}_cv)^T}{v^Tv} .$$

Now letting M_c and M_+ denote $(A\hat{D}_c)(A\hat{D}_c)^T$ and $(A\hat{D}_+)(A\hat{D}_+)^T$, respectively, and setting $w = A\hat{D}_cv$ and $\beta = v^Tv$, the Sherman-Morrison-Woodbury formula gives

$$M_+^{-1} = \left(M_c - \frac{ww^T}{\beta}\right)^{-1} = M_c^{-1} + \frac{M_c^{-1}ww^TM_c^{-1}}{\beta - w^TM_c^{-1}w} .$$

Setting $t = M_c^{-1}w$ and $\gamma = \beta - w^TM_c^{-1}w$, this is

$$= \left(I + \frac{tw^T}{\gamma}\right)M_c^{-1} .$$

In summary, at each step of the algorithm we save four vectors and two scalars : $v,u \in \mathbb{R}^n$, $w,t \in \mathbb{R}^m$, and $\beta,\gamma \in \mathbb{R}$ as follows :

$$\begin{aligned} v &= \left(\frac{(x_+-x_c)^Ty}{(\hat{D}_c^Ty)^T(\hat{D}_c^Ty)}\right)^{1/2}\hat{D}_c^Ty , & w &= A\hat{D}_cv , \\ u &= x_+ - x_c - \hat{D}_cv , & t &= [(A\hat{D}_c)(A\hat{D}_c)^T]^{-1}w , \\ \beta &= v^Tv , & \gamma &= \beta - w^Tt . \end{aligned} \tag{3.1.1}$$

The computation of $\hat{c}_p$ is carried out using

$$\hat{D}_k = D_0 + \frac{u_1v_1^T}{\beta_1} + \cdots + \frac{u_kv_k^T}{\beta_k} \tag{3.1.2}$$

and

$$[(A\hat{D}_k)(A\hat{D}_k)^T]^{-1} = \left(I + \frac{t_kw_k^T}{\gamma_k}\right)\cdots\left(I + \frac{t_1w_1^T}{\gamma_1}\right)[(AD_0)(AD_0)^T]^{-1}, \tag{3.1.3}$$

so that the only factorization needed for k steps of the algorithm is of the initial matrix $(AD_0)(AD_0)^T$. Notice also that vector or pipeline architectures are immediately applicable to the required computations.

We are indebted to M.J. Todd (private communication) for pointing out to us that computational savings may be realized if the updates to M^{-1} are saved in factored form. To this end, suppose that $(AD_c)(AD_c)^T = L_c L_c^T$. Then $[(AD_c)(AD_c)^T]^{-1} = L_c^{-T} L_c^{-1} \equiv N_c N_c^T$, and

$$M_+^{-1} = N_c N_c^T + \frac{N_c N_c^T w w^T N_c N_c^T}{\beta - w^T N_c N_c^T w} .$$

Setting $q = N_c^T w$ and $\hat{\gamma} = \beta - q^T q + [\beta(\beta - q^T q)]^{½}$, we obtain

$$\begin{aligned} M_+^{-1} &= N_c N_c^T + \frac{N_c q q^T N_c^T}{\beta - q^T q} \\ &= N_c \left(I + \frac{q q^T}{\beta - q^T q} \right) N_c^T \\ &= N_c \left(I + \frac{q q^T}{\hat{\gamma}} \right) \left(I + \frac{q q^T}{\hat{\gamma}} \right) N_c^T, \end{aligned}$$

so that the update to the Cholesky factor is given by

$$N_+ = N_c \left(I + \frac{q q^T}{\hat{\gamma}} \right) .$$

Using this approach, one would save, at each step of the algorithm, three vectors and two scalars : $v, u \in \mathbb{R}^n$, $q \in \mathbb{R}^m$, and $\beta, \hat{\gamma} \in \mathbb{R}$ as follows :

$$v = \left(\frac{(x_+ - x_c)^T y}{(\hat{D}_c^T y)^T (\hat{D}_c^T y)} \right)^{½} \hat{D}_c^T y , \qquad q = N_c^T A \hat{D}_c v ,$$

$$u = x_+ - x_c - \hat{D}_c v ,$$

$$\beta = v^T v , \qquad \hat{\gamma} = \beta - q^T q + [\beta(\beta - q^T q)]^{½} .$$

The computation of $\hat{c}_p$ would be carried out using (3.1.2) and

$$[(A\hat{D}_k)(A\hat{D}_k)^T]^{-1} = N_k N_k^T \tag{3.1.4}$$

where

$$N_k = N_0\left(I + \frac{q_1q_1^T}{\hat{\gamma}_1}\right)\left(I + \frac{q_2q_2^T}{\hat{\gamma}_2}\right)\cdots\left(I + \frac{q_kq_k^T}{\hat{\gamma}_k}\right).$$

Saving the updates in factored form would save storage of w , and the computation of q rather than t would save n floating point operations, where n is the number of nonzeros in the Cholesky factor of $(AD_0)(AD_0)^T$. The computation (3.1.4), however, would require mk more floating point operations than (3.1.3) at the kth step, so that the amount of computation saved would vary, depending on sparsity. The implementation for which we give numerical results in section 4 uses updates (3.1.1).

If the updating formula given by (2.17) supported by (2.16) is used, the computation of $\hat{c}_p$ requires twice as much storage and nearly twice as much work as this computation using updating formula (2.10) supported by (2.13). This is because the use of (2.17) leads to a rank-two update to obtain $(A\hat{D}_+)(A\hat{D}_+)^T$ from $(A\hat{D}_c)(A\hat{D}_c)^T$ at each step. At each step of the modified algorithm using the update (2.17), the eight vectors $s,v,u,t \in \mathbb{R}^n$ and $p,q,d,g \in \mathbb{R}^m$ and the four scalars β,γ,η, and ρ are saved :

$$\begin{aligned}
s &= x_+ - x_c, & p &= Au,\\
v &= \left(\frac{y^Ts}{(\hat{D}_c^{-T}s)^T(\hat{D}_c^{-T}s)}\right)^{1/2}\hat{D}_c^{-T}s, & q &= A\hat{D}_cs,\\
u &= v - \hat{D}_cy, & d &= M_c^{-1}\left(q + \frac{s^Ts}{\beta}p\right),\\
t &= \hat{D}_c^{-1}u, & g &= M_{c+1/2}^{-1}\,p,
\end{aligned}$$

$$\beta = y^T s, \qquad \eta = \beta + p^T d,$$
$$\gamma = v^T v, \qquad \rho = \beta + q^T g,$$

where $M_c = (A\hat{D}_c)(A\hat{D}_c)^T$,

$$M_{k-1}^{-1} = \left(I - \frac{g_{k-1}q_{k-1}^T}{\rho_{k-1}}\right)\left(I - \frac{d_{k-1}p_{k-1}^T}{\eta_{k-1}}\right) \cdots$$
$$\cdots \left(I - \frac{g_1 q_1^T}{\rho_1}\right)\left(I - \frac{d_1 p_1^T}{\eta_1}\right)[(AD_0)(AD_0)^T]^{-1} \quad ,$$

and

$$M_{k-\frac{1}{2}}^{-1} = \left(I - \frac{d_k p_k^T}{\eta_k}\right)\left(I - \frac{g_{k-1}q_{k-1}^T}{\rho_{k-1}}\right)\left(I - \frac{d_{k-1}p_{k-1}^T}{\eta_{k-1}}\right) \cdots$$
$$\cdots \left(I - \frac{g_1 q_1^T}{\rho_1}\right)\left(I - \frac{d_1 p_1^T}{\eta_1}\right)[(AD_0)(AD_0)^T]^{-1} \quad .$$

The approximation $\hat{D}_k$ is given by

$$\hat{D}_k = D_0 + \frac{u_1 s_1^T}{\beta_1} + \cdots + \frac{u_k s_k^T}{\beta_k} .$$

The computation of v requires $\hat{D}_c^{-T}s$. The matrix $\hat{D}_k^{-1}$ is

$$\hat{D}_k^{-1} = D_0^{-1} - \frac{t_1 v_1^T}{\gamma_1} - \cdots - \frac{t_k v_k^T}{\gamma_k} .$$

Using updating formula (2.17), it is still the case that the only matrix factorization needed for k steps of the algorithm is of the initial matrix $(AD_0)(AD_0)^T$.

3.2 Restarting strategy for the DMT-Karmarkar algorithm

In practice, there is a limit on the number of update vectors one is willing to store. We have implemented two ways of proceeding when the allotted storage has been used. The option we recommend is to restart, treating the current point as the initial point. This, of course, requires another matrix factorization. In our implementation, the restart strategy includes restarting when the approximation appears not to give a direction of sufficient decrease as well as when the maximum number of update vectors has been saved. We also plan to try a restart strategy based on the partial refactorization suggested by Karmarkar [1984] and studied by Shanno [1985b]. A second option that we have considered but do not recommend is to discard all previous updates, and treat the current point as the first point; that is, to replace the collection of updates with a single update as if the current point had been reached in one step from the initial point. One may also combine the two strategies, electing to perform an additional matrix factorization only when it is determined that progress is not being made. Using the pure strategy of discarding updates means that only the initial matrix is factored. This can yield the solution to the problem at modest expense and using very limited storage, but convergence in this case is not guaranteed. We recommend the restarting strategy over the discarding strategy, both for its sound theoretical basis and for its superior performance in practice. Use of the restarting strategy results in an algorithm that retains the polynomial worst case time bound of the Karmarkar algorithm, since our step acceptance criterion, to be discussed below, requires reduction of the potential function by at least the constant amount at each step that is guaranteed for the Karmarkar algorithm, and clearly one can always get this reduction by restarting.

A natural question to ask is whether it is advantageous to use update vectors at all in the DMT-Karmarkar algorithm. A reasonable alternative strategy within the framework we have established in (2.1) is, as mentioned by Gill, Murray, Saunders, and Wright [1986], to retain the same approximation to D for several iterations before recomputing a factorization. In an algorithm with periodic restarts, one would have $\hat{D}_i = D_0$ for $0 \leq i \leq k-1$ for some k, and then treat the kth iterate as the initial point. We have also explored the use of this strategy.

3.3 Linesearch and step acceptance criterion

Our implementation of the DMT-Karmarkar algorithm includes a linesearch with a three faceted step acceptance criterion as a way of selecting the steplength. The step acceptance criterion ensures that our algorithm with restarts retains the polynomial time bound of the unmodified Karmarkar algorithm but, rather than restricting steps to have a predetermined fixed length, allows taking longer steps when it appears advantageous to do so. We interpret failure to find an acceptable step after a specified number of trial steps as an indication that a restart is needed. Our first trial steplength is .99 of the distance to the edge of the simplex; our third trial steplength is .99 of the radius of the largest sphere that can be inscribed in the simplex, and our second trial steplength is midway between the first and third. Beginning with the fourth trial step, each is half as long as the previous trial step. Our step acceptance criterion is a combination of the Karmarkar criterion, the Goldstein-Armijo condition, and reduction of the linear objective function. The Karmarkar criterion, which is used to prove convergence of the algorithm in polynomial time, requires at least a constant amount of reduction in the potential function (2.3) at every step:

$$\begin{aligned} \tilde{f}\,(e - \alpha \hat{c}_p) &\leq \tilde{f}\,(e) - \delta \\ &= n \log c^T x_c - \delta, \end{aligned} \tag{3.3.1}$$

where δ is the minimum reduction in the potential function that is guaranteed for each step of the Karmarkar algorithm. The Goldstein-Armijo condition (see, for example, Dennis and Schnabel [1983]), requires that the average rate of decrease in the potential function from $\tilde{x}_c$ to $\tilde{x}_+$ be at least a prescribed fraction (λ) of the initial rate of decrease in that direction:

$$\tilde{f}\,(\tilde{x}_+) \leq \tilde{f}\,(\tilde{x}_c) + \lambda \nabla \tilde{f}\,(\tilde{x}_c)^T (\tilde{x}_+ - \tilde{x}_c)\ ;$$

that is,

$$\tilde{f}\,(e-\alpha\hat{c}_p) \le n \log c^T x_c - \alpha \frac{\lambda n}{c^T x_c} (Dc)^T \hat{c}_p \ . \tag{3.3.2}$$

In order to determine whether the trial step would result in a reduction of the linear function $c^T x$, we consider the objective value of the image of the trial step in the untransformed space, and determine whether the trial step satisfies

$$c^T x_+ = \frac{n}{e^T D\tilde{x}_+} c^T D\tilde{x}_+ < c^T x_c \ . \tag{3.3.3}$$

The lower bound δ on reduction in the potential function obtained by Karmarkar is dependent on the size of the problem as well as the steplength, and ensures reduction of the potential function at each step by at least .1 for a step in the direction $-c_p$ of length equal to one-fourth the radius of the largest sphere that can be inscribed in the simplex, provided the number of variables in the problem is at least 21. Since our test problems are larger than that, we use $\delta = .1$.

Our real objective is minimization of the linear function $c^T x$. While the linear function is not transformed to a linear function under projective transformation, straight lines do map to straight lines under projective transformation, so that the level sets of the linear function are mapped to flats in the transformed space. Therefore, if the direction we are considering gives descent on the linear function, it seems sensible to take the longest possible step in that direction that satisfies (3.3.1). In the event our direction is not a descent direction for the linear function, we guard against taking a step too long relative to the amount of reduction achieved in the potential function by requiring our step to satisfy (3.3.2) as well as (3.3.1). That is to say, we accept the step as soon as the trial step satisfies (3.3.1) and either (3.3.2) or (3.3.3). For an iteration at which a restart has been done, if a longer step does not satisfy (3.3.1) and either (3.3.2) or (3.3.3), we accept a step in the direction $-c_p$ of length equal to one-fourth the radius of the largest inscribed sphere. It will always satisfy (3.3.1).

4. Numerical results

Preliminary testing using an experimental implementation of the DMT-Karmarkar algorithm has given very encouraging computational results. We have used eight test problems. The first is a problem with one hundred seven variables and sixty-seven constraints that we obtained from Shell Development Company; the second through eighth are test problems from the Systems Optimization Laboratory at Stanford University that we obtained through netlib (see Gay [1985b]). The number of variables shown for each problem includes slack variables introduced to transform inequality constraints to equalities, but does not include the two additional slack variables discussed in Appendix I. Results obtained solving these problems using various options in the DMT-Karmarkar algorithm are summarized in tables 4.1 through 4.8. For each problem, a feasible starting point x_0 was obtained using the technique advocated by Karmarkar, which we include in Appendix I, from the initial point $(1, \cdots, 1)^T$. All options were then run from the same feasible starting point. The number of steps to convergence shown in the tables does not include steps required to obtain the starting point. The first option shown for each problem is restarting every step, which means that the Karmarkar algorithm, modified only by the incorporation of a linesearch, is used. In each instance, restart after k updates really means that a restart is done after saving **at most** k updates; that is, at least as often as every $k+1$ steps. A new factorization is done earlier whenever the linesearch is unsuccessful. Similarly, restart after k steps (using no updates, but retaining the same approximation to D between factorizations) means restart after **at most** k steps. To test the DMT-Karmarkar algorithm against the Karmarkar algorithm, we solved the test problems to only limited accuracy. We expect that relative times needed to compute more accurate solutions would be similar to those given in tables 4.1 through 4.8. For all problems, the stopping criterion was $c^T x \leq 10^{-3} c^T x_0$. This gave three to four digits of accuracy in the optimal objective values for all of the test problems except ISRAEL, where only one digit of accuracy in the optimal objective value was obtained. Subsequently, we continued the solution of ISRAEL. We obtained four digits of accuracy in the optimal objective value at the cost of twelve additional steps and factorizations using the Karmarkar algorithm, seventeen steps with nine factorizations restarting after one update, twenty-eight steps with seven factorizations restarting after three updates, and thirty-four steps with six factorizations restarting after five updates.

In tables 4.1 through 4.8 the update used is identified as follows :

update number	updating formula	righthand side of secant equation
1	(2.10)	y_b of (2.6)
2	(2.17)	y_b of (2.6)
3	(2.17)	y_p of (2.7)
4	(2.10)	y_p of (2.7)
5	(2.10)	y_t of (2.8)
6	(2.17)	y_t of (2.8)

For options involving the use of updates, we show only the results obtained using the most successful updating formula. In general, better performance was achieved using updating formulas 1, 4 and 5, the least change updates to $\hat{D}$ based on the barrier function, the potential function, and on the true D, respectively, than was achieved using updating formulas 2, 3 and 6, the updates that result from the strategy of least change to the Hessians (*i.e.* least change to $\hat{D}^{-1}$) of the functions used to obtain the approximations. The most successful updating formula was the simplest, the least change update to $\hat{D}$ based on the true D, identified as update 5. In cases where two or more updating formulas produced identical results in number of steps and factorizations needed to obtain the solutions, times given are for updating formula 5.

We have obtained timings solving the test problems on a Pyramid 90x, using the Unix operating system OSx version 2.5. Times are the sum of CPU times attributed to the user and to the operating system in seconds, rounded to the nearest second. The last column of each table contains normalized times, the ratio of the time required to obtain the solution using the given option to the time required using the Karmarkar algorithm (with linesearch).

For each of the test problems, use of the DMT-Karmarkar algorithm with periodic restarts, and using rank-one updates to approximate D at each step, results in obtaining the solution with (generally but not monotonically) progressively fewer matrix factorizations as the number of updates allowed between factorizations increases. Of course, the number of iterations required to obtain the solution increases as the number of factorizations decreases. Timings on the test problems indicate an overall reduction in the amount of work can be achieved using the DMT-Karmarkar algorithm, compared to the Karmarkar algorithm, for all problems except our smallest test problem, AFIRO (Table 4.2). Timings are accurate only to within about one second, so that little significance should be given to relative times for solving this very small problem that takes only two to three seconds. The largest percentages of savings in computational effort appear, as one would expect, in the larger problems, ISRAEL (Table 4.7) and BRANDY (Table 4.8).

The strategy of periodic restarts with no updates used may be advantageous if only a few steps are taken between restarts. The increase in number of steps required is generally greater, and the number of factorizations saved is generally smaller, than occurs when update vectors are used, but the computation of steps that require neither factorizations nor updates is very inexpensive.

Test problem: SHELL: 107 variables, 67 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		8	8	17	1.00
after 1 update	5	12	6	16	.94
after 2 updates	5	14	5	15	.88
after 4 updates	5	18	4	16	.94
after 9 updates	5	28	3	21	1.24
after 1 step		13	7	17	1.00
after 2 steps		19	7	19	1.12
after 3 steps		21	7	20	1.18

Table 4.1

Test problem: AFIRO: 51 variables, 27 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		7	7	2	1.00
after 1 update	1,4,5	11	6	3	1.50
after 2 updates	5	12	5	3	1.50
after 3 updates	5	16	4	3	1.50
after 7 updates	5	24	3	5	2.50
after 1 step		11	6	3	1.50
after 2 steps		16	6	3	1.50
after 3 steps		21	6	4	2.00

Table 4.2

Test problem: ADLITTLE: 138 variables, 56 constraints					
Option	**Update**	**Steps**	**Factorizations**	**Time**	**Time/Time(K)**
Restart					
every step		12	12	22	1.00
after 1 update	1	15	8	19	.86
after 2 updates	5	18	6	17	.77
after 4 updates	5	23	5	20	.91
after 7 updates	5	30	4	22	1.00
after 1 step		15	8	17	.77
after 2 steps		23	8	19	.86
after 3 steps		30	8	21	.95

Table 4.3

Test problem: SHARE2B: 162 variables, 96 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		9	9	26	1.00
after 1 update	5	14	7	24	.92
after 3 updates	5	22	6	28	1.08
after 6 updates	5	33	5	36	1.38
after 1 step		16	8	27	1.04
after 2 steps		22	8	29	1.12
after 5 steps		41	7	35	1.35

Table 4.4

Test problem: SHARE1B: 253 variables, 117 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		19	19	248	1.00
after 1 update	5	25	13	198	.80
after 2 updates	5	31	11	175	.71
after 3 updates	5	34	9	158	.64
after 4 updates	5	39	8	155	.63
after 5 updates	5	61	11	222	.90
after 6 updates	5	49	7	163	.66
after 9 updates	5	58	6	166	.67
after 1 step		28	14	193	.78
after 2 steps		37	13	192	.77
after 3 steps		44	12	190	.77

Table 4.5

Test problem: BEACONFD: 295 variables, 173 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		9	9	273	1.00
after 1 update	1,4,5	13	7	227	.83
after 2 updates	5	17	6	215	.79
after 4 updates	5	24	5	200	.73
after 10 updates	5	38	4	212	.78
after 20 updates	5	47	3	227	.83
after 1 step		13	7	222	.81
after 2 steps		19	7	232	.85
after 3 steps		24	7	241	.88

Table 4.6

Test problem: ISRAEL: 316 variables, 174 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		11	11	398	1.00
after 1 update	5	15	8	323	.81
after 2 updates	5	17	6	259	.65
after 3 updates	5	19	5	232	.58
after 7 updates	5	31	4	228	.57
after 15 updates	5	41	3	248	.62
after 1 step		16	8	326	.82
after 2 steps		22	8	332	.83
after 4 steps		30	7	312	.78

Table 4.7

Test problem: BRANDY: 292 variables, 182 constraints					
Option	Update	Steps	Factorizations	Time	Time/Time(K)
Restart					
every step		12	12	276	1.00
after 1 update	1,5	17	9	223	.81
after 2 updates	5	20	7	189	.68
after 3 updates	5	24	6	176	.64
after 6 updates	5	32	5	175	.64
after 10 updates	5	41	4	179	.65
after 1 step		20	10	244	.88
after 2 steps		25	9	229	.83
after 3 steps		33	9	240	.87

Table 4.8

Appendix I

The standard linear programming problem

$$\begin{array}{rl} \text{minimize} & \bar{c}^T\bar{x} \\ \text{subject to} & \bar{A}\bar{x} = b \\ & \bar{x} \geq 0, \end{array}$$

where $\bar{c},\bar{x} \in \mathbb{R}^{n-2}$, $b \in \mathbb{R}^{m-1}$, and $\bar{A} \in \mathbb{R}^{(m-1)\times(n-2)}$ can be transformed into the linear programming problem

$$\begin{array}{rl} \text{minimize} & c^Tx \\ \text{subject to} & Ax = 0 \\ & e^Tx = n \\ & x \geq 0, \end{array} \tag{A.1}$$

where $c,x,e \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $e=(1,\cdots,1)^T$ as follows. First, add a slack variable $\bar{x}_{n-1}$, express the requirement $\bar{A}\bar{x} = b$ as $\bar{A}\bar{x} = \bar{x}_{n-1}b$, and add a constraint requiring $\bar{x}_{n-1} = 1$. The equality constraints can now be stated as

$$\begin{bmatrix} & & & \\ & \bar{A} & & -b \\ & & & \\ 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \bar{x}_{n-2} \\ \bar{x}_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 1 \end{bmatrix}.$$

Now assume we have a bound B on the sum of the variables :

$$\sum_{i=1}^{n-1} \bar{x}_i \leq B$$

and introduce a second slack variable so that

$$\sum_{i=1}^{n} \bar{x}_i = B .$$

Using this condition, the constraint equation we have added can be written as

$$\bar{x}_{n-1} = \frac{1}{B} \sum_{i=1}^{n} \bar{x}_i \,.$$

Hence, the equality constraints have been put into homogeneous form:

$$A\bar{x} = \begin{bmatrix} & & & 0 \\ & & & \cdot \\ & \bar{A} & -b & \cdot \\ & & & \cdot \\ & & & 0 \\ 1 & \cdots & 1 \;\; (1-B) & 1 \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \bar{x}_{n-1} \\ \bar{x}_n \end{bmatrix} = \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 0 \end{bmatrix} .$$

Now scale the variables so that their sum is n : $x = \frac{n}{B}\,\bar{x}$. Since the number of variables has been increased by two, two zero components are appended to $\bar{c}$ to obtain $\hat{c}$.

If the optimal value of the objective function is not zero, but is known to be f^* , the objective function can easily be transformed to an objective function whose minimum value is zero. Minimizing $\hat{c}^T x$ is equivalent to minimizing $\hat{c}^T x - f^*$, and this second objective function is equivalent to

$$\hat{c}^T x - \frac{f^*}{n} \sum_{i=1}^{n} x_i \,,$$

so that

$$c = \hat{c} - \frac{f^*}{n}\, e \,. \tag{A.2}$$

For our test problems, the optimal value of the objective function was known, so that only the shift in c as above was necessary. If the minimum value of the objective function is not known, Karmarkar [1984] suggests the use of what he calls a sliding objective function. This involves attempting to solve the problem with objective function $c^T x$ with $c = \hat{c} - \frac{\hat{f}}{n}\, e$, where $\hat{f}$ is an estimate of f^* that is refined as the solution progresses. Todd and Burrell [1985] have suggested using the dual of problem (A.1) to obtain an estimate $\hat{f}$ that is a lower bound on f^* and that is refined at each iteration.

As noted by Karmarkar [1984], given any initial point $\hat{x} \in \mathbb{R}^n$, $\hat{x} > 0$, and assuming the linear programming problem is feasible, an initial feasible point for the linear programming problem (A.1) can be obtained by solving (for $x \in \mathbb{R}^{n+1}$) the problem

$$\begin{array}{rl} \text{minimize} & x_{n+1} \\ \text{subject to} & \hat{A}x = 0 \\ & e^T x = n + 1 \\ & x \geq 0, \end{array} \tag{A.3}$$

where $\hat{A} \in \mathbb{R}^{m \times (n+1)}$ has as its first n columns the matrix A and as its last column the vector $-A\hat{x}$. A feasible starting point for problem (A.3) is $(\hat{x}_1, \cdots, \hat{x}_n, 1)^T$.

REFERENCES

Kurt M. Anstreicher, "A monotonic projective algorithm for fractional linear programming," Yale School of Management (New Haven, CT, 1985, revised 1986). To appear in *Algorithmica* .

T.M. Cavalier and A.L. Soyster, "Some computational experience and a modification of the Karmarkar algorithm," Department of Industrial and Management Systems Engineering, The Pennsylvania State University (University Park, PA, 1985).

A. Charnes, T. Song, and M. Wolfe, "An explicit solution sequence and convergence of Karmarkar's algorithm," Center for Cybernetic Studies, The University of Texas at Austin, Research Report CCS 501 (Austin, TX, 1984).

J.E. Dennis, Jr. and Robert B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* (Prentice-Hall, Englewood Cliffs, NJ, 1983).

David M. Gay, "A variant of Karmarkar's linear programming algorithm for problems in standard form," A T & T Bell Laboratories Numerical Analysis Manuscript 85-10 (Murray Hill, NJ, 1985). To appear in *Mathematical Programming.*

David M. Gay, "Electronic mail distribution of linear programming test problems," *Committee on Algorithms Newsletter, Mathematical Programming Society* 13 (December, 1985).

Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright, "On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method," Systems Optimization Laboratory, Stanford University, Technical Report SOL 85-11 (Stanford, CA, 1985). To appear in *Mathematical Programming.*

Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright, "A note on nonlinear approaches to linear programming," Systems Optimization Laboratory, Stanford University, Technical Report SOL 86-7 (Stanford, CA, 1986).

Donald Goldfarb and Sanjay Mehrotra, "A relaxed version of Karmarkar's method," Department of Industrial Engineering and Operations Research, Columbia University in the City of New York (New York, NY, 1985, revised 1986).

N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4, (1984) 373-395.

Irvin J. Lustig, "A practical approach to Karmarkar's algorithm," Systems Optimization Laboratory, Stanford University, Technical Report SOL 85-5 (Stanford, CA, 1985).

D.F. Shanno, "A reduced gradient variant of Karmarkar's algorithm," Graduate School of Administration, University of California, Davis, Working Paper 85-01 (Davis, CA, 1985).

David F. Shanno, "Computing Karmarkar projections quickly," Graduate School of Administration, University of California, Davis, Working Paper 85-10 (Davis, CA, 1985).

David F. Shanno and Roy E. Marsten, "On implementing Karmarkar's method," Graduate School of Administration, University of California, Davis, Working Paper 85-01 (Davis, CA, 1985).

Michael J. Todd and Bruce P. Burrell, "An extension of Karmarkar's algorithm for linear programming using dual variables," School of Operations Research and Industrial Engineering, Cornell University, Technical Report No. 648 (Ithaca, NY, 1985). To appear in *Algorithmica* .

Michael J. Todd, Private communication (July, 1986).

J.A. Tomlin, "An experimental approach to Karmarkar's projective method for linear programming," Ketron, Inc. (Mountain View, CA, 1985). To appear in a *Mathematical Programming Study* on Computational Mathematical Programming.